

REMARKS

Claims 1 to 78 were pending in the application at the time of the advisory action. Claims 16, 18, 35, 37, 54, 56, 73, and 75 remain rejected for obviousness type double patenting. Claims 1 to 78 remain rejected as obvious.

Claims 16, 18, 35, 37, 54, 56, 73 and 75 stand rejected for obviousness-type double patenting in view of U.S. Patent No. 7,107,581, hereinafter referred to as the '581 patent.

Claim 18 in the instant application stands rejected in view of Claim 12 in the '581 patent, while Claim 16 in the instant application stands rejected in view of Claims 53, 54 of the '581 patent.

Claim 18 in the instant application depends from Claims 1, 16 and 17. Accordingly, Claim 18 includes all the limitations of Claims 1, 16, 17 and 18. Claim 12 in the '581 patent depends from Claims 1, 11 and 12. Thus, Claim 12 includes all the limitations of Claims 1, 11 and 12.

Below is a claim chart showing a comparison between Claims 18 and Claim 12. Within the claim chart is text that is included with parentheses and that is underlined, this text points out differences between the claims. The differences pointed out are not intended to be inclusive of all differences, but are sufficient to demonstrate that the obviousness-type double patenting rejection is not well founded.

Claim 12, '581 Patent	Claim 18 instant application
(Claim 1) receiving a first instruction defined for a first processor having a first base, said instruction comprising an operator and at least one operand having an operand type;	(Claim 1) <u>(No receiving operation recited. No consideration of first processor base recited.)</u>

<p><u>(No validating operation recited. No input stack considerations recited in particular an input stack representing the state of operand stack. No input instruction of said first instruction recited.)</u></p>	<p>validating at least one input stack associated with a first instruction configured to operate on at least one operand of a first type, each of said at least one input stack associated with an input instruction of said first instruction, each input stack representing the state of an operand stack associated with an input instruction upon execution of said input instruction</p>
<p>converting said first instruction to a second instruction optimized for a second processor having a second base when overflow is not possible based at least in part on said operator and the relationship between said operand type and said second base, said second base smaller than said first base; and</p>	<p>converting said first instruction to a second instruction configured to operate on at least one operand of a second type, said second type smaller than said first type, said converting based at least in part on the relative size of said first type and said second type, wherein said second instruction is different from said first instruction; and  <u>(No consideration of overflow, second processor base, or operator characteristics recited.)</u></p>
<p>converting instructions in a chain of instructions to a wider base when said at least one operand carries the potential for overflow beyond said second base and when said operator is sensitive to overflow, said chain bounded by said second instruction and a third instruction that is the source of potential overflow associated with said at least one operand, said third instruction having been previously optimized, said wider base larger than said second base and smaller or equal to said first base</p>	<p><u>(No converting of instructions in a chain to a different base recited. No consideration of a wider base based on overflow recited and no requirement for a previously optimized third instruction recited.)</u></p>
<p><u>(No matching of operand types, no use of input stack, and no changing type of</u></p>	<p>matching said second type with an operand type of at least one operand in said at</p>

<p><u>instruction based on operand type recited.)</u></p>	<p>least one input stack associated with said second instruction, said matching comprising changing the type of instructions in a chain of instructions to equal said second type if said operand type is less than said second type, said chain bounded by said second instruction and a third instruction that is the source of said at least one operand</p>
<p>(Claim 11) further comprising recording conversion results, said recording comprising: determining potential overflow associated with said second instruction; and generating an output stack based at least in part on execution of said second instruction</p>	<p>(Claim 16) further comprising recording conversion results, said recording comprising: determining potential overflow associated with said second instruction; and generating an output stack based at least in part on execution of said second instruction</p>
<p><u>(No elaboration of determining including indicating second instruction has potential for overflow as recited in first indicating operation and coupling that with second indicating.)</u></p>	<p>(Claim 17) said determining further comprises: indicating said second instruction has potential overflow if said second type does not equal said first type, if said second instruction does not remove potential overflow, and if said second instruction creates potential overflow; and indicating said second instruction has potential overflow if said second type does not equal said first type, if said second instruction does not remove potential overflow, if said second instruction does not create potential overflow, if said second instruction propagates potential overflow, and</p>

	if at least one operand in said at least one input stack has potential overflow
<p>(Claim 12)  said determining further comprises:</p> <p>indicating said second instruction has potential overflow if said second type does not equal said first type, if said second instruction does not remove potential overflow, and if overflow is possible based at least in part on the type of said second instruction and the relationship between said first type and said second type; and</p> <p>indicating said second instruction has potential overflow if said second type does not equal said first type, if said second instruction does not remove potential overflow, if overflow is not possible based at least in part on the type of said second instruction and the relationship between said first type and said second type, if said second instruction propagates potential overflow, and if at least one operand in said at least one input stack has potential overflow</p>	<p>(Claim 18)  said determining further comprises:</p> <p>indicating said second instruction has potential overflow if said second type does not equal said first type, if said second instruction does not remove potential overflow, and if overflow is possible based at least in part on the type of said second instruction and the relationship between said first type and said second type; and</p> <p>indicating said second instruction has potential overflow if said second type does not equal said first type, if said second instruction does not remove potential overflow, if overflow is not possible based at least in part on the type of said second instruction and the relationship between said first type and said second type, if said second instruction propagates potential overflow, and if at least one operand in said at least one input stack has potential overflow.</p>

As previously stated, the rejection reduces the claims to a gist and ignores explicit claim limitations. For example, Claim 17 in the instant application recites two limitations, which for purposes of discussion will be considered as "a & b",

where "a" is the first indicating limitation and "b" is the second indicating limitation in Claim 17. Claim 18 in the instant application also recites two limitations "c & d". A comparison of the "if" conditions in elements a, b, c, and d is presented in the following table.

Claim 17		Claim 18	
a	b	c	d
if said second type does not equal said first type	if said second type does not equal said first type	if said second type does not equal said first type	if said second type does not equal said first type
if said second instruction does not remove potential overflow; and	if said second instruction does not remove potential overflow	if said second instruction does not remove potential overflow	if said second instruction does not remove potential overflow
if said second instruction creates potential overflow	if said second instruction propagates potential overflow;		if said second instruction propagates potential overflow
	if said second instruction does not create potential overflow and	if said second instruction does not remove potential overflow	
	if at least one operand in said at least one input stack has potential overflow		if at least one operand in said at least one input stack has potential overflow
		if overflow is possible based at least in part on the type of said second instruction and the relationship between said first type and said second	

		type	
			if overflow is not possible based at least in part on the type of said second instruction and the relationship between said first type and said second type

This unambiguously demonstrates that the combination of limitations "a & b" is different from the combination of limitations of elements "c & d" and by definition Claim 18 includes both "a & b" and "c & d". The claims are not the intermingling of all the elements as rejected, but rather Claim 17 includes a limitation "a" and a limitation "b" that are both different from limitations "c" and "d" of Claim 18. Claim 12 of the '581 patent includes only limitations "c & d."

The rejection glosses over the fact that Claim 17 is separate and distinct from Claim 18 by stating:

Although '581 claim 12 does not recite 'indicating said second instruction overflow. . . (as recited in instant claim 17), claim 18 and claim 17, together, can be addressed together with the subject matter of '58 1 claim 12. That is, the matter recited in instant claim 17 amounts to a slight variation of --yet equivalent to-- the subject matter of instant claim 18; hence would be treated as subsumed under claim 18 and therefore obvious when treating it with the '581 claim 12, as set forth above ( see \*).

It has been demonstrated that limitation "a" is not equivalent to any limitation in Claim 18 and limitation "b" is not equivalent to any limitation in Claim 18. Therefore, Claim 17 is not equivalent to Claim 18 as asserted. Mixing and recombining the separate limitations results in different

limitations than those recited in these Claims. Claim 17 checks for a set of conditions separate and distinct from those of Claim 18 as demonstrated above. Accordingly, the rejection demonstrates that explicit claim limitations have been ignored and reduced to a gist that is incorrect and not supported by the claim language. This alone is sufficient to overcome the rejection.

Next the rejection extracts a part of validating limitation of Claim 18 and stated:

*'581 claim 12 does not recite 'one input stack associated with a first instruction configured to operate on at least one operand of a first type, each of said at least one input stack associated with an input instruction of said first instruction, each input stack representing the state of an operand stack associated with an input instruction upon execution of said input instruction'; but in view of '581 claim 12 reciting of 'determining potential overflow associated with said second instruction and generating an output stack based on execution of said 2<sup>nd</sup> instruction (see '581 claim 11) . . . and indicating said 2<sup>nd</sup> instruction. . . has potential overflow, and if at least one operand . . . one input stack has potential overflow (see '581 claim 12), the above limitation is considered analogously disclosed because output stack with operand being modified entails input stack operands potential overflow problem, according to the above from '581 to accomplish the above conversion operating on the operands associated with an input stack.*

Again no teaching or suggestion of a validating operation has been alleged. Rather, the claim is selectively dissected and then reduced to "the above conversion operating on the operands associated with an input stack." There has been no citation or assertion of any validating limitation. Further, the rejection extracts one condition used to determine whether to indicate the second instruction has potential overflow in the '581 patent and then uses logic, which has nothing to do with the validating operation, based an output stack of the '581 patent. Claim 18 includes an input stack and an operand

stack and defines the relationship between the two with respect to the validating. Claim 18 also expressly defines that the input instruction is part of the first instruction and not the second instruction relied upon in the rejection. The rejection extracts unrelated pieces of Claim 16 in the '581 patent and ignores the express relationships and the operations recited in Claim 18 of the instant application. This also is sufficient to overcome the rejection.

Moreover, as demonstrated in the above claim chart, to arrive at Claim 18 in the instant application requires numerous modifications to Claim 12 of the '581 patent. Many of which are not addressed in the rejection.

In summary, as previously pointed out Claim 18 includes the validating operation of Claim 1 and includes four different indicating operations when the claim is considered as a whole as demonstrated in detail above.. Therefore, to render Claim 18 obvious, Claims 1, 11 and 12 of the '581 patent must suggest each of the elements of Claim 18. "All Claim Limitations Must Be Taught or Suggested." MPEP § 2143.03, 8<sup>th</sup> Ed., Rev. 5, pg. 2100-131 (August 2005).

There has been no citation of any suggestion of a validating operation in Claims 1, 11 and 12 of the '581 operation. This cannot be dismissed because to do so fails to consider Claim 18 as a whole, which the MPEP requires. Similarly, the four distinct indicating operations of Claim 18 cannot be morphed into something different from the explicit claim limitations, because again this fails to consider Claim 18 as a whole. Similar comments are also applicable to each of Claims 37, 56 and 75. Applicants respectfully request reconsideration and withdrawal of the obviousness-type double patenting rejection of each of Claims 18, 37, 56 and 75 in view of the '581 patent.

Claim 16 in the instant application depends from Claim 1. Accordingly, Claim 16 includes all the limitations of Claims 1



and 16. Below is a claim chart showing a comparison between Claims 16 and Claim 53 of the '581 patent. Within the claim chart is text that is included with parentheses and that is underlined, this text points out differences between the claims. The differences pointed out are not intended to be inclusive of all differences, but are sufficient to demonstrate that the obviousness-type double patenting rejection is not well founded.

Claim 53, '581 Patent	Claim 16 instant application
A method of using an application software program including arithmetic expression optimization of at least one instruction targeted to a processor, the method comprising:	(Claim 1) A method for arithmetic expression optimization, comprising:
receiving the software program on said processor, said software program optimized according to a method comprising:	<u>(No receiving of optimized software program recited.)</u>
receiving a first instruction defined for a first processor having a first base, said instruction comprising an operator and at least one operand having an operand type;	<u>(No receiving operation recited. No consideration of first processor base recited.)</u>
<u>(No validating operation recited. No input stack considerations recited in particular an input stack representing the state of operand stack. No input instruction of said first instruction recited.)</u>	validating at least one input stack associated with a first instruction configured to operate on at least one operand of a first type, each of said at least one input stack associated with an input instruction of said first instruction, each input stack representing the state of an operand stack associated with an input instruction upon execution of said input instruction;
converting said first instruction to a second	converting said first instruction to a second

<p>instruction optimized for a second processor having a second base when overflow is not possible based at least in part on said operator and the relationship between said operand type and said second base, said second base smaller than said first base;</p>	<p>instruction configured to operate on at least one operand of a second type, said second type smaller than said first type, said converting based at least in part on the relative size of said first type and said second type, wherein said second instruction is different from said first instruction;  <u>(No consideration of overflow, second processor base, or operator characteristics recited.)</u></p>
<p>converting instructions in a chain of instructions to a wider base when said at least one operand carries the potential for overflow beyond said second base and when said operator is sensitive to overflow, said chain bounded by said second instruction and a third instruction that is the source of potential overflow associated with said at least one operand, said third instruction having been previously optimized, said wider base larger than said second base and smaller or equal to said first base; and</p>	<p><u>(No converting of instructions in a chain to a different base recited. No consideration of a wider base based on overflow recited and no requirement for a previously optimized third instruction recited.)</u></p>
<p><u>(No matching of operand types, no use of input stack, and no changing type of instruction based on operand type recited.)</u></p>	<p>matching said second type with an operand type of at least one operand in said at least one input stack associated with said second instruction, said matching comprising changing the type of instructions in a chain of instructions to equal said second type if said operand type is less than said second type, said chain bounded by said second instruction and a third instruction that is the source of said at least one operand</p>
<p>executing said at least one instruction on said processor.</p>	

(No recording conversion results and no generating an output stack recited.)

(Claim 16)  
further comprising  
recording conversion results,  
said recording comprising:  
determining potential  
overflow associated with said  
second instruction; and  
generating an output  
stack based at least in part  
on execution of said second  
instruction

The rationale for continuing the rejection, as best it is understood, stated in part:

Applicants have submitted that claim 53 of '581 must suggest each limitation of claim 16; and that the Rejection fails to provide such suggestion. The level of one of ordinary skill in the art has it that what appears to not explicit can be interpreted as being strongly implied; and such implied teaching can serve a fuel as to a suggestion leading to more possibilities based on the existence of the more explicit teachings in the claims.

If the rejection is continued, the Examiner is respectfully requested to provide more specificity with respect to the level of skill in the art and support for such a level of skill. General knowledge concerning the Java language and a smart card for example fails to provide a basis for the very detailed and different modifications that must be made to Claim 53 to arrive at Claim 16 in the instant application.

Further, the rejection must demonstrate that after the numerous modifications are made to Claim 53 of the ' 581 patent that Claim 53 would still work for its intended purpose. There has been no showing that Claim 53 would still work for its intended purposes when the consideration of the various bases of the different processors in Claim 53 is eliminated. There has been no showing that Claim 53 would work with other than the recited third instruction that was previously optimized. There has been no showing that Claim 53 would still work for its intended purpose when the considerations of the operator

are eliminated. The rejection has failed to consider the numerous modifications that are required and the ramifications based on those modifications based solely on Claim 53 and the level of skill in the art.

Further, the rationale relied upon with respect to Claim 18 to find the input stacks recited within the validating operation of Claim 16 is not found in Claim 53. Therefore, for numerous reasons, a prima facie obviousness rejection still has not been made. Claim 16 includes an input stack and an operand stack and defines the relationship between the two with respect to the validating. Claim 16 also expressly defines that the input instruction is part of the first instruction. The rejection has cited no teaching or suggestion of such elements in Claim 16. This also is sufficient to overcome the rejection.

Again no teaching or suggestion of a validating operation has been alleged. To the extent that the rejection of Claim 16 relies upon the rejection of Claim 18, the above discussion of Claim 1 with respect to Claim 18 is incorporated herein by reference.

Moreover, as demonstrated in the above claim chart, to arrive at Claim 16 in the instant application requires numerous modifications to Claim 53 of the '581 patent. Many of which are not addressed in the rejection

Only one of these distinctions is sufficient to overcome the obviousness rejection. Claim 54 includes limitations similar to those of Claim 53 and so the remarks with respect to Claim 53 are directly applicable to Claim 54. Similar comments are also applicable to each of Claims 35, 54 and 73. Applicants respectfully request reconsideration and withdrawal of the obviousness-type double patenting rejection of each of Claims 16, 35, 54 and 73 in view of the '581 patent.

Claims 16, 18, 35, 37, 54, 56, 73 and 75 stand rejected for obviousness-type double patenting in view of U.S. Patent No. 7,207, 037, hereinafter referred to as the '037 patent.

Claim 18 in the instant application stands rejected in view of Claim 12 in the '581 patent, while Claim 16 in the instant application stands rejected in view of Claims 53, 54 of the '581 patent.

Claim 18 in the instant application depends from Claims 1, 16 and 17. Accordingly, Claim 18 includes all the limitations of Claims 1, 16, 17 and 18. Claim 12 in the '037 patent depends from Claims 1, 11 and 12. Thus, Claim 12 includes all the limitations of Claims 1, 11 and 12.

Below is a claim chart showing a comparison between Claims 18 and Claim 12. Within the claim chart is text that is included with parentheses and that is underlined, this text points out differences between the claims. The differences pointed out are not intended to be inclusive of all differences, but are sufficient to demonstrate that the obviousness-type double patenting rejection is not well founded.

Claim 12, '037 Patent	Claim 18 instant application
(Claim 1) receiving a first instruction defined for a first processor having a first base, said instruction comprising an operator and at least one operand;	(Claim 1) <u>(No receiving operation recited. No consideration of first processor base recited.)</u>
<u>(No validating operation recited. No input stack considerations recited in particular an input stack representing the state of operand stack. No input instruction of said first instruction recited.)</u>	validating at least one input stack associated with a first instruction configured to operate on at least one operand of a first type, each of said at least one input stack associated with an input instruction of said first instruction, each input stack representing the state of an

	operand stack associated with an input instruction upon execution of said input instruction
converting said first instruction to a second instruction optimized for a second processor having a second base when said at least one operand does not carry potential overflow beyond said second base or when said operator is insensitive to overflow, said second base smaller than said first base;	converting said first instruction to a second instruction configured to operate on at least one operand of a second type, said second type smaller than said first type, said converting based at least in part on the relative size of said first type and said second type, wherein said second instruction is different from said first instruction;; and <u>(No consideration of overflow, second processor base, operator characteristics, or operand overflow characteristics recited.)</u>
converting instructions in a chain of instructions to a wider base when said at least one operand carries the potential for overflow beyond said second base and when said operator is sensitive to overflow, said chain bounded by said second instruction and a third instruction that is the source of potential overflow associated with said at least one operand, said third instruction having been previously optimized, said wider base larger than said second base and smaller or equal to said first base.	<u>(No converting of instructions in a chain to a different base recited. No consideration of a wider base based on overflow recited and no requirement for a previously optimized third instruction recited.)</u>
<u>(No matching of operand types, no use of input stack, and no changing type of instruction based on operand type recited.)</u>	matching said second type with an operand type of at least one operand in said at least one input stack associated with said second instruction, said matching comprising changing the type of instructions in a chain of instructions to equal said second type if said operand type is less than said second

	type, said chain bounded by said second instruction and a third instruction that is the source of said at least one operand
(Claim 11) further comprising recording conversion results, said recording comprising: determining potential overflow associated with said second instruction; and generating an output stack based at least in part on execution of said second instruction	(Claim 16) further comprising recording conversion results, said recording comprising: determining potential overflow associated with said second instruction; and generating an output stack based at least in part on execution of said second instruction
(Claim 12) wherein said determining further comprises: indicating said second instruction has potential overflow if said second type does not equal said first type, if said second instruction does not remove potential overflow, and if said second instruction creates potential overflow; and indicating said second instruction has potential overflow if said second type does not equal said first type, if said second instruction does not remove potential overflow, if said second instruction does not create potential overflow, if said second instruction propagates potential overflow, and if at least one operand in said at least one input stack has potential overflow	(Claim 17) said determining further comprises: indicating said second instruction has potential overflow if said second type does not equal said first type, if said second instruction does not remove potential overflow, and if said second instruction creates potential overflow; and indicating said second instruction has potential overflow if said second type does not equal said first type, if said second instruction does not remove potential overflow, if said second instruction does not create potential overflow, if said second instruction propagates potential overflow, and if at least one operand in said at least one input stack has potential overflow
(No elaboration of determining including indicating second instruction has potential for overflow as	(Claim 18) said determining further comprises: indicating said

recited in first indicating operation and coupling that with second indicating.)

second instruction has potential overflow if said second type does not equal said first type, if said second instruction does not remove potential overflow, and if overflow is possible based at least in part on the type of said second instruction and the relationship between said first type and said second type; and

indicating said second instruction has potential overflow if said second type does not equal said first type, if said second instruction does not remove potential overflow, if overflow is not possible based at least in part on the type of said second instruction and the relationship between said first type and said second type, if said second instruction propagates potential overflow, and if at least one operand in said at least one input stack has potential overflow.

Since the rejection of Claim 18 in view of the '037 patent relies completely upon the rejection of Claim 18 in view of the '581 patent, the above comments with respect to Claim 18 and the '581 patent are incorporated herein by reference. Similar comments are also applicable to each of Claims 37, 56 and 75. Applicants respectfully request reconsideration and withdrawal of the obviousness-type double patenting rejection of each of Claims 18, 37, 56 and 75 in view of the '037 patent.



Claim 16 in the instant application depends from Claim 1. Accordingly, Claim 16 includes all the limitations of Claims 1 and 16. Below is a claim chart showing a comparison between Claims 16 and Claim 53 of the '037 patent. Within the claim chart is text that is included with parentheses and that is underlined, this text points out differences between the claims. The differences pointed out are not intended to be inclusive of all differences, but are sufficient to demonstrate that the obviousness-type double patenting rejection is not well founded.

Claim 53, '037 Patent	Claim 16 instant application
A method of using an application software program including arithmetic expression optimization of at least one instruction targeted to a processor, the method comprising:	(Claim 1) A method for arithmetic expression optimization, comprising:
receiving the software program on said processor, said software program optimized according to a method comprising:	<u>(No receiving of optimized software program recited.)</u>
receiving a first instruction defined for a first processor having a first base, said instruction comprising an operator and at least one operand;	<u>(No receiving operation recited. No consideration of first processor base recited.)</u>
<u>(No validating operation recited. No input stack considerations recited in particular an input stack representing the state of operand stack. No input instruction of said first instruction recited.)</u>	validating at least one input stack associated with a first instruction configured to operate on at least one operand of a first type, each of said at least one input stack associated with an input instruction of said first instruction, each input stack representing the state of an operand stack associated with an input instruction upon execution of said input instruction;

<p>converting said first instruction to a second instruction optimized for a second processor having a second base when said at least one operand does not carry potential overflow beyond said second base or when said operator is insensitive to overflow, said second base smaller than said first base;</p>	<p>converting said first instruction to a second instruction configured to operate on at least one operand of a second type, said second type smaller than said first type, said converting based at least in part on the relative size of said first type and said second type, wherein said second instruction is different from said first instruction;  <u>(No consideration of overflow, second processor base, operator characteristics, or operand overflow characteristics recited.)</u></p>
<p>converting instructions in a chain of instructions to a wider base when said at least one operand carries the potential for overflow beyond said second base and when said operator is sensitive to overflow, said chain bounded by said second instruction and a third instruction that is the source of potential overflow associated with said at least one operand, said third instruction having been previously optimized, said wider base larger than said second base and smaller or equal to said first base; and executing said at least one instruction on said processor.</p>	<p><u>(No converting of instructions in a chain to a different base, no consideration of a wider base based on overflow recited and no requirement for a previously optimized third instruction recited.)</u></p>
<p><u>(No matching of operand types, no use of input stack, and no changing type of instruction based on operand type recited.)</u></p>	<p>matching said second type with an operand type of at least one operand in said at least one input stack associated with said second instruction, said matching comprising changing the type of instructions in a chain of instructions to equal said second type if said operand type is less than said second type, said chain bounded by said second instruction and a</p>

	third instruction that is the source of said at least one operand
executing said at least one instruction on said processor.	
<u>(No recording conversion results and no generating an output stack recited.)</u>	(Claim 16) further comprising recording conversion results, said recording comprising: determining potential overflow associated with said second instruction; and generating an output stack based at least in part on execution of said second instruction

Since the rejection of Claim 16 in view of the '037 patent relies completely upon the rejection of Claim 16 in view of the '581 patent, the above comments with respect to Claim 16 and the '581 patent are incorporated herein by reference. Claim 54 includes limitations similar to those of Claim 53 and so the remarks with respect to Claim 53 are directly applicable to Claim 54. Similar comments are also applicable to each of Claims 35, 54 and 73. Applicants respectfully request reconsideration and withdrawal of the obviousness-type double patenting rejection of each of Claims 16, 35, 54 and 73 in view of the '037 patent.

Claims 1 to 78 stand rejected under 35 U.S.C. 103(a) as being unpatentable over U.S. Patent No. 5,740,441, hereinafter referred to as Yellin, in view of U.S. Patent No. 6,308,317, hereinafter referred to as Wilkinson.

Applicant respectfully notes that the MPEP requires that a reference be considered as a whole, "A prior art reference must be considered in its entirety." MPEP § 2143.02, VI., 8<sup>th</sup> Ed., Rev. 5, pg. 2100-124 (August 2005). Therefore, even though there is a combination of references, each reference must support the interpretation that forms the basis of the rejection. Yellin fails to do this.

With all due respect, the rejection mischaracterizes the background and the teaching of Yellin. It is well established that the bytecodes of the Java programming language used in Yellin are computer platform independent and so are independent of any underlying computer processor architecture etc. The purpose of Yellin is not as asserted in the rejection to change code to operate on different underlying platform architectures, but rather as stated by Yellin:

The present invention verifies the integrity of computer programs written in a bytecode language, commercialized as the JAVA bytecode language, which uses a restricted set of data type specific bytecodes. All the available source code bytecodes in the language either (A) are stack data consuming bytecodes that have associated data type restrictions as to the types of data that can be processed by each such bytecode, (B) do not utilize stack data but affect the stack by either adding data of known data type to the stack or by removing data from the stack without regard to data type, or (C) neither use stack data nor add data to the stack.

The present invention provides a verifier tool and method for identifying, prior to execution of a bytecode program, any instruction sequence that attempts to process data of the wrong type for such a bytecode or if the execution of any bytecode instructions in the specified program would cause underflow or overflow of the operand stack, and to prevent the use of such a program.  
(Emphasis Added)

Yellin, Col. 1, line 57 to Col. 2, line 7.

The issue is to verify the integrity of a bytecode computer program that has been transferred over a network, such as the Internet, that includes heterogeneous computer systems and not to modify or change the bytecode program. Yellin specifically describes this fact:

For many purposes, particularly the integrity of downloaded computer programs, the Internet is a "hostile environment." A downloaded bytecode program may contain errors involving the data types of operands not matching the data type restrictions of the instructions using those

operands, which may cause the program to fail during execution. Even worse, a bytecode program might attempt to create object references (e.g., by loading a computed number into the operand stack and then attempting to use the computed number as an object handle) and to thereby breach the security and/or integrity of the user's computer.

Use of the bytecode verifier 120 in accordance with the present invention enables verification of a bytecode program's integrity and allows the use of an interpreter 122 which does not execute the usual stack monitoring instructions during program execution, thereby greatly accelerating the program interpretation process.

Yellin, Col. 5, line 54 to Col. 6, line 3.

Yellin is only concerned with verifying that the bytecodes of the downloaded computer program will execute properly and does this by verifying the bytecodes based on known characteristics of the bytecode. Specifically,

The interpreter, prior to executing any bytecode program, executes a bytecode program verifier procedure that verifies the integrity of a specified program by identifying any bytecode instruction that would process data of the wrong type for such a bytecode and any bytecode instruction sequences in the specified program that would cause underflow or overflow of the operand stack.

Yellin, Abstract, lines 3 to 9.

Further, the overflow and underflow of Yellin are associated with the stack and not arithmetic overflows and underflows as implied in the rejection. A stack underflow or overflow is associated with pushing and popping data with respect to a stack and not with the type of a particular operand as recited in the Claims. Yellin is concerned with "any bytecode instruction sequences in the specified program that would cause underflow or overflow of the operand stack." (Yellin, Abstract).

Yellin does not suggest "adding data and rearranging stack calling structures relative to discrepancies of first and second types." Yellin is only concerned with stopping a bytecode program from being executed by a bytecode interpreter, as expressly stated by Yellin. In addition, Yellin expressly teaches away from modification of the code by stating:

If the program verifier finds any instructions that violate predefined stack usage and data type usage restrictions, execution of the program by the interpreter is prevented.

Yellin, Abstract, lines 9 to 12.

It is not just the execution of the offending bytecode that is prevented, but rather execution of the entire program. The statements by Yellin are clear complete and unambiguous.

In spite of the consistent teaching of Yellin the verifier only verifies the bytecode program and kills execution if a problem is found, the rejection dissects the reference, takes teachings out of context and uses the dissected out of context teachings to mischaracterize the reference.

For example, the rejection stated:

However, Yellin discloses adding data and rearranging stack calling structure relative to discrepancies of first and second type ( see col. 1, lines 60-67; col. 20, lines 24-35; col. 21, lines 12-37; step 394, Fig. 4B) hence has suggested modifying the runtime instruction as verified by the stack to accommodate data being ported from a larger size operand --or higher platform-- to a smaller size operand - or lower platform ( see col. 5, lines 14-64; *multiple computer platforms, underlying instruction sets-* col. 1 , lines 7- 14). (Emphasis in original)

Examination below of each of the cited sections in Yellin demonstrates that Yellin has been misinterpreted. Col. 1 lines 60 to 67 of Yellin stated:

All the available source code bytecodes in the language either (A) are stack data consuming bytecodes that have associated data type restrictions as to the types of data that can be processed by each such bytecode, (B) do not utilize stack data but affect the stack by either adding data of known data type to the stack or by removing data from the stack without regard to data type, or (C) neither use stack data nor add data to the stack.

This section of Yellin describes the bytecodes of the JAVA bytecode language and as noted above, it is well known that such bytecodes are computer platform independent. This section provides no support for the assertion that Yellin is "adding data and rearranging stack calling structure relative to discrepancies of first and second type."

Col. 20, lines 24 to 35 in context of lines 23 to 42 recite:

```
/* Update jsr bit vector array */
If the current instruction is in a subroutine that is the target of a jsr
{
  For each level of jsr applicable to the current instruction
  {
    Update corresponding jsr bit vector to indicate register(s) accessed or
    modified by the current instruction
    /* Set of "marked" registers can only be increased, not decreased */
  }
}
/* Update all affected SnapShots and changed bits */
Determine set of all successor instructions, including:
(A) the next instruction if the current instruction is not an unconditional
goto, a return, or a throw,
(B) the target of a conditional or unconditional branch,
(C) all exception handlers for this instruction,
(D) when the current instruction is a return instruction, the successor
instructions are the instructions immediately following all jsr's that
target the called subroutine.

If the program can "fall off" the last instruction
{ Set VerificationSuccess to False
  Return with Abort return code value }
/*
```

This section of pseudocode describes the bytecode verifier and not the bytecode program being verified. Specifically, it

is concerned with updating bits associated with "a "jsr" bit vector array 306 that stores zero or more bit vectors associated with the zero or more subroutine calls required to reach the instruction currently being processed" (Yellin, Col. 6, lines 19 to 22) and updating a snapshot array of the verifier.

Accordingly, this portion of Yellin has nothing to do with modification to the bytecode program as asserted in the rejection. The program code for the bytecode verifier teaches or suggests nothing with respect to "adding data and rearranging stack calling structure relative to discrepancies of first and second type" in the bytecode program being verified.

Similarly, Col. 21, lines 12 to 37 of Yellin are pseudocode for elements of the bytecode verifier used to verify the bytecode program and not the bytecode program being verified. Changing a Snapshot in the instruction verifier as taught in step 394 of Fig. 4B again teaches what the verifier does in verification process and fails to teach or suggest anything with respect to "adding data and rearranging stack calling structure relative to discrepancies of first and second type" in the bytecode program being verified by Yellin. Thus, as previously stated, the rejection mischaracterizes the teaching of Yellin and relies upon this mischaracterization to assert that Applicants' claims are obvious.

Claim 1 recites that a first instruction is converted to obtain a second instruction and defines a property of both instructions. The bytecode verifier of Yellin does not convert any instruction, but as noted above verifies that a bytecode in a downloaded bytecode program has the proper configuration for executing properly. The claim further defines the result of the conversion in that "said second type smaller than said first type" and defines a requirement for the conversion, "said converting based at least in part on the relative size of said



first type and said second type." Thus, the claim expressly recited the input to the converting, the first instruction; the output of the converting, the second instruction; and the converting utilizes at least the relative size of the two types. A verifier that determines the correct configuration of a bytecode and prevents execution of that program if the configuration is incorrect is totally unrelated to such a conversion process.

Yellin must suggest starting with a first instruction and after conversion ending up with a second instruction with the two instructions having properties with the relationship recited. Further, Yellin must suggest that the conversion is based at least on the relative size of said first type and said second type.

Verification of bytecodes and aborting when a problem is encountered, as taught by Yellin, fails to suggest or disclose an operation that starts with a first instruction and generates a second instruction as recited in Claim 1. Applicants note that a second reference was cited with respect to "changing the type of instructions in a chain," but assuming the combination is correct, the second reference fails to correct the deficiencies of the primary reference noted above and so the combination fails to render Claim 1 obvious. Applicants respectfully request reconsideration and withdrawal of the obviousness rejection of Claim 1.

Claims 2 to 19 depend from Claim 1 and so distinguish over the combination of references for at least the same reasons as Claim 1. Applicants request reconsideration and withdrawal of the obviousness rejection of each of Claims 2 to 19.

Claims 20, 39, 58, 77 and 78 each include limitations similar to those of Claim 1. Accordingly, the above remarks with respect to Claim 1 are applicable to each of these claims and are incorporated herein by reference. Applicants request

reconsideration and withdrawal of the obviousness rejection of each of Claims 20, 39, 58, 77 and 78.

Claims 21 to 38 depend from Claim 20 and so distinguish over the combination of references for at least the same reasons as Claim 20. Applicants request reconsideration and withdrawal of the obviousness rejection of each of Claims 21 to 38.

Claims 40 to 57 depend from Claim 39 and so distinguish over the combination of references for at least the same reasons as Claim 39. Applicants request reconsideration and withdrawal of the obviousness rejection of each of Claims 40 to 57.

Claims 59 to 76 depend from Claim 58 and so distinguish over the combination of references for at least the same reasons as Claim 58. Applicants request reconsideration and withdrawal of the obviousness rejection of each of Claims 59 to 76.

Claims 1 to 78 remain in the application. For the foregoing reasons, Applicant(s) respectfully request allowance of all pending claims. If the Examiner has any questions relating to the above, the Examiner is respectfully requested to telephone the undersigned Attorney for Applicant(s).

**CERTIFICATE OF MAILING**

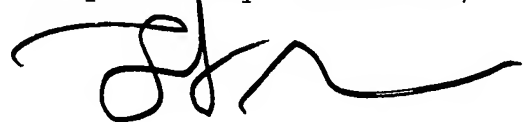
I hereby certify that this correspondence is being deposited with the United States Postal Service with sufficient postage as first class mail in an envelope addressed to: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450, on December 21, 2007.



Attorney for Applicant(s)

December 21, 2007  
Date of Signature

Respectfully submitted,



Forrest Gunnison  
Attorney for Applicant(s)  
Reg. No. 32,899  
Tel.: (831) 655-0880